REMARKS

In the Office Action dated July 1, 2009, the Examiner:

- rejected claims 6, 8, 114-124 under 35 USC § 101 as non-statutory subject matter; and
- rejected claims 1-6, 8, 22-33, 41-43, 51-96 and 114-128 under 35 USC § 103(a) as obvious over the combination of the 1996 article by Crespo and Bier entitled: *WebWriter: A Browser-Based Editor for Constructing Web Applications* ("WebWriter I") and the 1997 article by Crespo, Chang and Bier entitled *Responsive Interaction for a Large Web Application: The Meteor Shower Architecture in the WebWriter II Editor* ("WebWriter II").

Applicant believes that the Patent Office is mistaken regarding the nature and scope of applicant's invention and the cited prior art, and for some claims, has failed to establish a *prima facie* case for obviousness. The cited WebWriter Editor does not operate in a WYSIWYG mode with respect to editing of dynamic parts of pages, which only display as placeholders during editing, not as computed or functional content. There is compelling evidence of that mode of operation in figure 7b of the prior art.

Neither of the cited WebWriter articles describes or suggests running edited applications during editing. WebWriter I has two page generators, one built into the WebWriter editor for displaying pages during editing, and a second one called "the WebWriter page generator" for running an application after editing it. Applicant's invention, however, has a single page generator for both running the application during editing and later at runtime.

Applicant's claims recite these and other distinctions over the cited combination in various ways, as further explained below. This response specifically adds section A.iii discussing WebWriters page generators and section C and E discussing examiners arguments of the most recent office action dated July 1st, 2009, section C discussing independent and section E dependent claims.

Applicant has amended claim 22 and 59 in order to clarify them and claim 125 to avoid futher 101 issues and Applicant submits that all pending claims are patentable over the cited references for the reasons detailed below.

I. ARGUMENTS

A. THE EXAMINER IS MISTAKEN REGARDING THE NATURE AND SCOPE OF THE CITED PRIOR ART

i. Overview

The Patent Office asserts that the combination of the WebWriter I article and the WebWriter II article makes Applicant's claims obvious. However, Applicant respectfully disagrees, and continues to believe that the Patent Office is mistaken regarding the nature and scope of the disclosure in the WebWriter articles. The WebWriter I article describes a software development system consisting of two programs: the WebWriter editor and the WebWriter page generator. Further, the program named WebWriter Page Generator is a separate program unrelated to editing. The examiner assumes that the WebWriter Editor uses the WebWriter Page Generator to generate and display pages while editing, however nothing in the references supports that position. The WebWriter page generator is just the page generator of the WebWriter development system and the WebWriter editor has its own page generation (see section A.iii).

The editor described in WebWriter I is mainly concerned with editing the static parts of a page, and shows dynamic parts as placeholders during editing (*See* Figure 7b). For example, the WebWriter I article states that "*that static parts of each page are created in advance by the designer using a text editor*" (*see* p. 2, 2nd para. under heading "The WebWriter Application Model") and further, "*WebWriter then adds a placeholder for the output area [which] will be replaced* ***at runtime*** *by the output of a script . . . .*" (*see* page 4, 2nd para. under heading "Placing Dynamic Areas", emphasis added).

In contrast, Applicant's invention is concerned with editing components, which are dynamic parts. Further, Applicant's invention actually displays the executed dynamic parts during editing, and except for the addition of editing features, e.g., handles, the appearance and function of the content during editing is virtually the same as at run-time. This is achieved by running the edited web application during editing. In contrast, WebWriter I runs the edited application only after editing, as discussed further below, and makes no teaching or suggestion of running an application during an editing operation.

Applicant's Claims express these distinctions by stating, for example, that (1) the edited application is running during editing, or (2) the component displayed during editing has a similar appearance as on the generated page with the addition of editing features or (3) components cooperate with the editor during editing.

For example, claims 1 and 59 require a document generator that runs the application during editing, claim 90 requires that scripts in the edited document remain running during editing, and method claim 125 explicitly requires a running step during editing. Claim 26 requires that the documents be similar at run time and during editing, to wit: "*at least a part of the second document appears and functions similar to the first document.*" Claims 74 and 114 require components that cooperate with the editor during editing.

Section ii below contains a more detailed discussion explaining why the WebWriter articles do not teach or suggest running applications during editing. Section iii describes the separate nature of the two programs, the WebWriter Editor and the WebWriter Page Generator. Section iv discusses the internal architecture of WebWriter and explains why it is not even possible or feasible to run the application during editing when using that architecture. Section viii discusses components cooperating with the editor during editing.

ii. The Examiner Misinterprets WebWriter

In the current Rejection, the Examiner asserts that WebWriter I is capable running the edited web application while editing, which is incorrect. Although the WebWriter system has the WebWriter Page Generator for running an application, this page generator is just used after editing while running an application, not during editing. The editor uses its own page generation part for showing pages during editing, which does not execute the edited application. The examiner seems to simply assume that the WebWriter Page Generator is the one used during editing, but the WebWriter articles do not support that position, and the Examiner does not provide any reasoning or support for that position. This is discussed this in detail in section A. iii. below

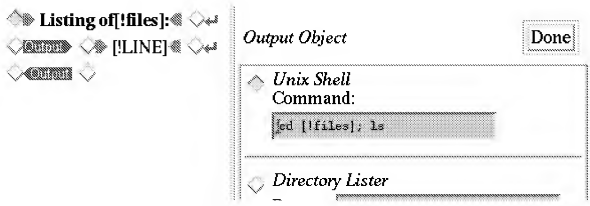
WebWriter I follows a strict separation of editing time and run time. During editing, WebWriter I only shows the static parts of a template page while dynamically generated parts are displayed as placeholders. Upon completion of editing, the template page is saved. At runtime, the dynamically generated parts are executed and their output is displayed. This is explained as

noted in Section i above, and also on p. 6, col. 2 of WebWriter I, in the section entitled "Running the Application:" "***Once*** *a Web Writer application is created and* ***saved*** *to disk it can be* ***run*** *in one of two ways*" (emphasis added)..

As a result, the output generated by scripts at run time generally looks very different than the placeholders displayed while editing, so that pages displayed by WebWriter during editing and at runtime usually look very different as well. For example, as shown below, Fig. 7(b) of the WebWriter I reference shows the editing view and Fig. 6(b) shows the runtime view of a single page template implementing a file listing application. The runtime view correctly shows a list of files, while the editing view shows a placeholder consisting of handles surrounding the output formatting specification *"[!Line]."*

Listing of /project/girweb/figs/coins/:
coins.htm
dime.gif
nickel.gif
penny.gif
quarter.gif

WebWriter Fig. 6(b) –View while Running



WebWriter Fig. 7(b) –View while Editing

Also, Fig. 10(b) and Fig. 11 show the difference between the runtime and editing views in WebWriter I of a page template implementing a walkie talkie application. The runtime view in Fig. 10(b) shows the text of a message received by the walkie talkie while the editing view just shows the placeholder, namely *"[!Line]"* surrounded by handles.

iii. The WebWriter Page Generator and the Page Generation inside the WebWriter Editor are separate

WebWriter is a system for web application development having two distinct environments: a development time environment (the "WebWriter Editor") and a run-time environment (the "WebWriter Page Generator"). For example, the abstract of WebWriter I says:

> "This paper describes WebWriter an integrated system for constructing Web Applications . . . Webwriter includes a direct manipulation Web page editor, the *WebWriter Editor* . . . and the *WebWriter Page Generator* , which generates new pages as the application runs."

Both programs are described in a section called "Implementation," which is split into two subsections: "The WebWriter Editor" and "The WebWriter Page Generator." (see pp. 8-9). The WebWriter editor and WebWriter page generator are clearly separate programs, the editor being used during editing and the page generator being used at run time. The first paragraph of the section entitled "WebWriter Page Generator" on page 9 explains that the Web Writer Page Generator is used at run-time, i.e. while an application created with WebWriter is running: "**When** a WebWriter application is **running**, each new page is assembled by the **WebWriter Page Generator**, **another** server based CGI C++ **program**..." (emphasis added) The use of the word "**another**" makes clear that these are indeed separate programs.

There is no explicit teaching in the article, much less a hint or suggestion, that the WebWriter Editor is using the WebWriter Page Generator. On the contrary, the sub-section called "The WebWriter Editor" describes the tree-based page generation method used by the editor (see pp. 8-9). The 2nd paragraph of the section says "*Each page **generated by the editor** contains HTML code that represents the appearance of the current document in the left column.*" During editing, the document is stored in the form of a content tree, and "*after executing each user specified action, WebWriter generates the next page by traversing the content **tree . . . .***" (page 8, last paragraph, emphasis added). This explains precisely and explicitly how page

templates are displayed during editing, i.e., by traversing the content tree, and not by using the WebWriter Page Generator.

The examiner has specifically mentioned the phrase "on the fly" which is used in the implementation section of the WebWriter I page generator, but this phrase certainly does not mean what the examiner suggests. On page 9 column 1, the WebWriter I article states: "*The Page Generator generates a new page on the fly as it scans through the template page,*" and this just means that the WebWriter Page Generator is implemented using the "on the fly" technique, which is a common technique to implement parsers: It processes the document character by character and while doing so processes dynamic areas as soon as they occur in the input i.e. on-the-fly "*As it reads it copies each character to its output stream until it encounters ...* " (page 9 column 1, last full paragraph)The sentence explains the inner workings of the WebWriter Page Generator, but does not express in any way that the page generator is running during editing.

For all these reasons it becomes clear that the WebWriter Page Generator runs after editing and not during editing. Since the WebWriter Page Generator executes the edited application and the scripts of dynamic areas (as detailed in "The WebWriter Page Generator" section on page 9) it becomes clear that this also happens after editing and not during editing. And vice versa it is clear that editing features are produced only by the document generation inside the WebWriter Editor (as detailed in "The WebWriter Editor"on page 8) and not by the WebWriter Page Generator.

With respect to WebWriter II, the examiner only refers to this article for establishing prior art for "server" or "web server" on page 3 of the recent office action. The examiner does not suggest that WebWriter II discloses running dynamic web run applications during editing.

iv. Editor Architecture

The editors described in the WebWriter articles and applicant's editor have very different architecture. The classical editors described WebWriter I and WebWriter II have load and save operations. When loading a document, it is transformed into a content tree. Editing operations modify the tree. Finally, on a save, the tree is unparsed and stored into a file. During editing after each operation, the tree is traversed and displayed to the user. (See page 8 column 2 Headline

The WebWriter Editor). In addition, there is the WebWriter Page Generator that, after the edited page has been saved, reads a file and executes scripts. This classical architecture requires that applications be executed only when editing is finished and the tree is sent to the server and saved to a file, because the file is needed for execution. During editing, documents are displayed inside the editor, i.e., inside an environment that considerably differs from the runtime environment, e.g., URLs, URL parameters, and the page generator are different. In order to successfully run an application, however, the document must run in environment that closely resembles the runtime environment.

In contrast, applicant's editor runs the application being edited in a first browser window and the client part of the editor in another window. The user interacts with the application running in the first window as usual. It runs in its normal runtime environment, e.g., using its normal URLs, URL parameters etc. and also uses its normal page generator. Therefore, it appears fully functional. While editing, the page generator adds additional editing features to the generated pages, in a way that does not interfere with the normal operation of the application. The editing features include handles. The user can click on a handle or interact with the editor window to issue a modification. Then, the client and server part of the editor perform the modification directly in the document template file. Afterwards, the editor asks the browser to reload the page in the first window, which causes the now modified application to be run by the page generator as usual.

v. The Combination of WebWriter I and WebWriter II Is Improper

In rejecting applicant's claims, the examiner has relied on two prior art references describing two variants of the WebWriter Editor referred to herein as WebWriter I and WebWriter II. WebWriter II is a newer development from the same authors trying to solve the same problems as WebWriter I, so one might assume that WebWriter II already contains as far as possible all the features of WebWriter I.

WebWriter II, however, uses a different architecture called "meteor shower" which performs most of the editing operations on the client computer (except for loading the document before editing and saving the document after editing), while WebWriter I did most of the editing operations on the server. (*See* WebWriter II, sec. 1.2 entitled "Increasing interactive performance"). This new architecture makes WebWriter II work faster but teaches away from

applicant's idea because running the edited server side application necessarily takes place on the server. Applicant's editor does the editing in a distributed way, since many operations are done client side (e.g. displaying information about components and running client side parts of an edited document during editing) while others are done server side (e.g. modifying the edited document and running the server side parts of it during editing).

Because of the different architectures used in WebWriter I and WebWriter II, applicant submits that it is not obvious to simply combine WebWriter I and WebWriter II as suggested by the Examiner. Including a feature from one version into the other requires adaptation to another architecture (e.g., moved from server to client or vice versa), which may not be trivial or even possible. To pick and choose features from different prior art disclosures in order to form an obviousness rejection amounts to hindsight reasoning, which is impermissible. Applicant therefore submits that the combination is improper, and on that basis, applicant submits that the Examiner has failed to establish a *prima facie* case of obviousness.

vi. The Examiner Fails to Provide Reasoning for All Independent Claims

On page 3 of the current action, the Examiner lumps all of the independent claims together, then describes his assertion that WebWriter I discloses a page generator program and an editor program as claimed. However, while that basis for rejection may be applicable to independent claims 1 and 59, the language and structure of independent claims 6, 22, 26, 51, 114 and 125 are much different than that of claims 1 and 59. Further, the Examiner did not consider any differences in claim language or scope, nor did the Examiner provide any detailed reasoning that would support rejection of these claims of different language and scope, and therefore, the Examiner has failed to make a prima facie case for obviousness of independent claims 6, 22, 26, 51, 114 and 125.

Further, while the examiner does provide some additional reasoning for claims 74 and 90, for all the other independent claims, he still refers to his reasoning for claim 1. Applicant responds to these arguments claim by claim in section C.

vii. Components

Applicant's invention allows the developer create web applications by plugging together software components on document templates using the inventive editor. These special type of

components contain instructions, are executable on the server side, and dynamically generate browser code. In applicant's preferred embodiment, components are implemented as objects of an object oriented programming language.

Unfortunately, the current rejection does not specify what is considered prior art for components. With respect to claim 2, the first claim mentioning the terms "components" and "document templates," the examiner refers to page 2, column 1 of WebWriter I, see Heading "*The WebWriter Application Model*" and refers to the term "*template page*". Therefore, applicant assumes that WebWriter's template pages are alleged to be prior art for the claimed page templates. The WebWriter I article defines dynamic areas as locations within a template page ("*template page, which includes the specification of locations within the page, called dynamic areas . . . .*") and adds that a script is specified for each dynamic area. Applicant therefore assumes that the examiner interprets scripts specified for the dynamic areas as prior art for the components recited in applicant's claims.

The cited section of WebWriter I also states that the scripts are run at run-time to provide computed content for the dynamic areas ("*dynamic areas, where computed content should be inserted at* ***run-time*** *. . . a script that will be run to provide to provide the computed content for that dynamic area*" (emphasis added)).

viii. Components may Cooperate with the Editor

Just as for WebWriter's scripts of dynamic areas, Applicant's components are able to run at run-time. However, applicant's components are run and used during editing as well, and this feature is not taught or suggested in the cited combination. In fact, applicant's components cooperate with the editor during editing in various ways, e.g. by generating browser code for display of the component inside the editor, by drawing handles, or by collecting and providing information about the component to the editor.

In contrast, the scripts specified for WebWriter's dynamic areas are not used at all during editing, but just replaced by placeholders. ("*WebWriter then adds a* ***placeholder*** *for the output area at the current insert location point")*

In fact, applicant submits that a further distinctive feature is recited by the claim language "components cooperating with the editor during editing." The present Office Action does not cite any specific portion of WebWriter as disclosing this feature, because of course it is not disclosed.

WebWriter's dynamic areas call independent Unix scripts at runtime, as explained on p. 9 at the end of column 1 and top of column 2: "*When WebWriter encounters an OUTPUT tag it ... runs the specified program on its arguments ...*". The WebWriter Editor cannot be extended by any external components. This is explained in WebWriter I on page 5 column 1 Section entitled "Script" "*The user selects from a small set of* ***built-in*** *modules ...*" (emphasis added). It is only the WebWriter Page generator that can call arbitrary scripts and more specifically scripts of dynamic areas at run time.

This distinction is recited in the claim language of claim 74 ("*the components having the ability to cooperate with the editor*") and of claim 114 ("*the components including first features adapted to cooperate with an editor in dynamically editing said component.*")

ix. Nested Components

WebWriter's dynamic areas are represented using a special <output> tag as explained on page 9 column 1. The output tag contains a formatting string. As explained at the end of column 1 and top of column 2 of page 9 "*When WebWriter encounters an OUTPUT tag it extracts the SCRIPT field and formatting string, runs the specified program on its arguments, formats the results using the formatting string and writes the result to the output.*". There is no teaching or motivation that the formatting string could contain itself nested OUTPUT tags.

In his argument with respect to claim 4 on page 28, the examiner cites a page template containing a component as a nested component. However, page templates are not components in the sense of claim 4 (or 81 and 87) because base claim 2 (or 74) requires page templates containing components. So, only items that can be placed on page templates are seen as components in the sense of claim 2, and page templates cannot contain other page templates, so page templates are not components in the sense of the claim.

In contrast, applicant's components can be nested. Inside a component on a document template, HTML code and other components can be denoted. An enclosing component can generate browser code for insertion at the begin tag and the end tag and it can also dynamically control insertion of content into the final document. For example, the component can hide its content, insert it once, or multiple times. Thus, the components nested inside may dynamically be hidden, or multiple instances of a component may be displayed. In this way, the number and

kind of components on a generated page is no longer static but can dynamically change at run time.

This is denoted using the claim language of independent claim 74: "*the set of components on the generated documents can vary for different document requests for said document template;*" and, for example, dependent claim 4: "*wherein at least one of the components contains at least one other component;*" and finally claim 8: "*a component is nested within a component*".

Note that not only do applicant's components have these features, but applicant's editor also has the ability to handle components with these features and, for example, has operations to place one component inside another one, or display multiple instances of a component during editing.

x. Components Interactively react on User Input

Some of Applicant's components can not only display themselves during a first browser request, but also on a subsequent request, after user interaction, the components can process the data sent back from the browser to the server. These components are called interactive. Several dependent claims are based on interactive components.

In contrast, the scripts called by WebWriter's dynamic areas are just used for processing of a single request. As explained on p. 9 of WebWriter I, left column, Heading The Web Writer Page Generator, the scripts are called during document generation to produce output for an output area. This is taking place during processing of a single request. The scripts do not have a concept of multiple requests or subsequent requests and can therefore also not react in any way on a subsequent request.

This feature is recited for example in claim 24:"*at least one of the components … can react on subsequent document requests containing user responses by executing selected instructions of said component.*"

B. THE INDEPENDENT CLAIMS ARE PATENTABLE OVER THE COMBINATION OF WEBWRITER I AND WEBWRITER II

Claim 1

Claim 1 is an independent claim that is directed to a basic two-part structure: a document generator that runs a web application and an editor. It requires including editing features in generated documents and the editor program operating via the editing features. The claim does not rely on components.

As explained in sections A.ii and A.iii the WebWriter Editor does not run the edited application during editing. In stark contrast, applicant's editor does. Claim 1 is considered patentable over the cited combination at least because (1) the recited document generator runs at least part of the edited application by generating documents that include editing features, and (2) the recited editor operates via editing features incorporated into said generated documents. This is recited in Claim 1 as:

> a document generator program running at least part of one of the applications being edited and generating the generated documents, said generated documents including additional editing features for interpretation by the browser program; and
> an editor program dynamically operating on the generated documents displayed by the browser program via the editing features.

The recent office action refers generally to the description of the WebWriter Page Generator in the WebWriter I article (see Office Action dated Sep. 17, 2008 at p. 3), but provides no specific reasoning to support the rejection. There is no teaching or suggestion in either WebWriter article that the WebWriter Page Generator inserts any editing features into the generated pages. In fact, this would not make any sense because the WebWriter Page Generator does not run during editing – it runs only after editing, as explained in sections A.ii and A.iii. The "on the fly" phrase used in the prior art article and specifically cited by the examiner just specifies the internal working of the page generator, e.g., at runtime, the placeholders are replaced with the dynamic content, and the article does not teach or suggest running the WebWriter Page Generator during editing.

In addition, the phrases "editor program operating on the generated documents" and "via the editing features" are key structural limitations of the claim language that were not specifically addressed by the Examiner, and which connect the running application to the editing process. The claim recites that the editing features are part of the generated documents, and the preamble states that the generated documents are displayed by the browser. The editor is then coupled via the editing features, and *voila*, you are editing a running application in the browser.

The examiner cited WebWriter II only to incorporate the explicit teaching of server computer and web server. This, however, has no influence on the working of the WebWriter Page Generator and thus, Claim 1 is patentable over the cited combination.

The examiner also states "see page 2, first paragraph for buttons and computing content on the fly for components as claimed in claim 59." This comment is irrelevant for claim 59 as well as for claims 1, 26 and 90, since these claims do not recite components and do not have any limitations on components. Furthermore, buttons work on client side and are therefore not server side components. In contrast, for example, claim 6 requires components being executed on the server and claim 22 requires components being executed by the page generator. The term "on the fly" is not used in the cited portion of the article on page 2. Applicant does, however, discuss this reference in his arguments with regards to claim 6.

Claim 6

As explained in section A.ii and A.i, WebWriter replaces dynamic parts of pages with placeholders during editing. The placeholders show the script name and a formatting string and so usually look very different from the appearance of a component on a generated page. In contrast, claim 6 has been amended to require that the components have a similar appearance as on the generated page. Therefore, applicant submits that the claim is patentable over the cited combination.

The examiner did not include specific reasoning for the rejection of claim 6, but instead relied upon his reasoning as applied to claim 1. However, Applicant submits that this reasoning is simply not applicable to claim 6, in part, because claim 6 recites components. Further, the failure to provide detailed reasoning amounts to a failure to make a *prima facie* case of obviousness.

Further, the cited buttons work on client side and are therefore not server side components. In contrast, claim 6 requires components being executed on the server. Therefore, buttons are not components in the sense of claim 6.

Claim 22

Applicant amended claim 22 to clarify that the editor indeed operates on the documents generated by the page generator and by executing the components.

As explained in section A.ii and A.iii the WebWriter Editor does not execute part of the edited application during editing. As described in WebWriter I page 9 the WebWriter scripts of dynamic areas are executed only after editing by the WebWriter Page Generator. In contrast claim 22 requires an editor operating on documents generated by the document generator that has instructions for executing components and so requires that the components are executed during editing. Therefore applicant submits that the claim is patentable over the cited combination.

The examiner did not include specific reasoning to support his rejection of claim 22, but relied on his reasoning as applied to claim 1. Applicant submits that this reasoning is not applicable because the claim has been amended as shown above, and also because claim 22 includes the recitation of components while claim 1 does not. Since the Examiner has failed to provide detailed reasoning, he has not made a *prima facie* case of obviousness.

Claim 26

Claim 26 describes a system to modify a first document stored on a server whereby a second document is displayed to the user that appears and **functions** similar to the first document. Applicant emphasizes the fact the recited elements of the claim are located on the server, wherein the server comprises:

> a document store;
>
> a first software program including instructions for transforming at least one first document into a second document having features which permit editing of the first document such that at least a part of the second document appears and functions the same as the first document; and
>
> "*a second software program including . . .* instructions *to modify the first document,*

Claim 26 stands rejected on the basis of a combination of WebWriter I and WebWriter II. As explained in detail in section A.ii and A.iii WebWriter does not execute the application being edited during editing. Therefore it is clear that the application being editing does not function at all during editing. In contrast the claim requires "*at least a part of the second document appears and* ***functions*** *similar to the first document*."

In fact WebWriter transforms a first document being edited into a second document which permits editing of the first, thereby trying maintain the appearance of the static parts of the document. However, as discussed in section A.i, WebWriter does not make any effort to maintain the appearance of the dynamically generated parts of the document or of the functionality of the document.

The examiner did not include specific reasoning to support his rejection of claim 26, but relied on his reasoning as applied to claim 1. Because the Examiner has failed to provide detailed reasoning, he has not made a *prima facie* case for obviousness.

Therefore, Claim 26 is patentable over the cited combination.

Claim 51

Claim 51 is an independent claim describing a system for editing components on web document templates. As explained in section A.viii, the WebWriter Editor does not use the scripts associated with the dynamic areas. These scripts are used only after editing, at run-time. In contrast, claim 51 has been amended to require components having features to cooperate in editing the component. Therefore, applicant submits that the claim is patentable over the cited combination.

The examiner did not include specific reasoning for claim 51 but relied on his reasoning of claim 1. Applicant submits that this reasoning is not applicable because the claim has been amended as shown above, and also because claim 51 includes the recitation of components and of features to cooperate in editing the component while claim 1 does not. Because the Examiner has failed to provide detailed reasoning, he has not made a *prima facie* case for obviousness.

Claim 59

Claim 59 is an independent claim directed to a software development system for dynamic web documents.

As explained above, the inventive editor is capable of running a dynamic web document being developed during editing and so the dynamic web document appears functional during editing. This distinction is expressed by the following claim language, "*generated documents ... which look and function similar to the end user's view of the documents"* and by *"the editor program comprising first instructions for requesting the document generator to process a dynamic web document leading to a generated document*." This language makes clear that the editor indeed runs the dynamic web document during editing, by requesting the document generator to process a dynamic web document. Additional claim language, namely "*instructions to modify* ***the dynamic web document***" make clear that the dynamic web document that is being edited is the one being requested.

The examiner did not include specific reasoning for claim 59 but relied on his reasoning as applied to claim 1, wherein the examiner refers to the WebWriter Page Generator. The WebWriter Page Generator, however, runs after editing as previously explained, and so it is clear that the WebWriter Page Generator does not receive requests from the WebWriter Editor during editing. In contrast, the claim requires *"the editor program comprising first instructions for requesting the document generator to process a dynamic web document leading to a generated document*." For all these reasons, applicant submits that claim 59 is patentable over the cited references.

In his reasoning about claim 1 the examiner added a comment "also see page 2, first paragraph for buttons and computing content on the fly for components as claimed in claim 59". Since claim 59, like claim 1, does not recite any components, applicant does not believe this is applicable for claim 59.

Claim 74

Claim 74 is an independent claim describing a software development system for document templates and stands rejected as obvious based on the combination of WebWriter I and WebWriter II. In the Rejection, the Examiner did not include specific reasoning for claim 74 but handled claim 74 together with claim 1. Because the Examiner has failed to provide detailed reasoning, he has not made a *prima facie* case for obviousness.

Nevertheless claim 74 requires "*wherein the set of components on the generated documents can vary for different document requests*" and "*the components having the ability to cooperate with the editor*". Claim 1 does not require "components". Therefore, applicant submits that the examiners reasoning given is not applicable and therefore irrelevant for claim 74. As explained in sections A.viii and A.vii, WebWriter I and WebWriter II do not teach or suggest components cooperating with the editor. In contrast, claim 74 explicitly requires "*the components having the ability to cooperate with the editor*".

The claim explicitly requires that the set of components on the generated documents can vary for different document requests for the same document template. In WebWriter, each dynamic area is filled by its script exactly once per request, as explained in section "The Web Writer Page Generator" on page 9 of the WebWriter I article. It is not possible that a script associated with a dynamic area is excluded from generation or generated multiple times and so

the number of scripts of dynamic areas executed per template page is fixed. In contrast, the claim requires that the set of components on the generated documents can vary for different document requests. For all these reasons, applicant submits that claim 74 is patentable over the cited references.

Claim 90

Independent Claim 90 is directed to an editor embodiment. The editor is used with a web browser and allows a user to edit a document being actively displayed by the browser "*wherein scripts contained in said document remain functional during editing.*" Further, the editor includes a client part, e.g., a first software program for execution within the browser that initiates editing functions when the user clicks on the displayed document.

The Examiner did not include specific reasoning for claim 90 but included claim 90 with his reasoning for claim 1. Because the Examiner has failed to provide detailed reasoning, he has not made a *prima facie* case for obviousness.

As explained in sections A.ii and A.iii above, WebWriter does not run the application during editing. In contrast, claim 90 requires scripts contained in the document to remain functional during editing.

In addition, claim 90 recites a first software program for execution within the browser. WebWriter I does not have such a program, since it does not seem to generate pages with client side scripts. WebWriter II does have client side scripts, however, as explained in section A.v above, the combination of WebWriter I and WebWriter II is improper since they employ different architectures.

For all these reasons, applicant submits that claim 90 is patentable over the cited combination.

Claim 114

Independent Claim 114 is directed to an editor embodiment, but was rejected without any specific discussion in the grouping of claims described on page 3 of the recent Office Action. For this reason, applicant submits that the Examiner failed to make a prima facie case for obviousness.

Claim 114 is an independent claim describing a system for editing components on web document templates. Claim 114 requires "*the components including first features adapted to*

cooperate with an editor in dynamically editing said component." Claim 1 does not include such a feature, it does not even require "components". Therefore applicant submits that the examiners reasoning given is not applicable and therefore irrelevant for claim 114.

Claim 114 requires "*the components including first features adapted to* ***cooperate*** *with an editor"* (emphasis added). As reasoned in detail in section A.viii above, WebWriter does not teach or suggest components that cooperate with the editor. Applicant submits that therefore claim 114 is patentable over the cited combination.

Also, despite the Examiner's comments to the contrary in the advisory action a part of the editor cannot be interpreted as a component because the part lacks first features to cooperate with the editor in editing this part itself as required by the claim language "*components including first features to cooperate with an editor in editing said component.*" In addition, the use of the word cooperate makes clear that the components are not part of the editor.

Claim 125

Claim 125 is an independent claim describing a method useful for editing an application that is built using components and that operates by generating documents. The claim stands rejected as obvious based on the combination of WebWriter I and WebWriter II. In the Rejection, the Examiner did not include specific reasoning for claim 125 but handled the claim together with claim 1. For this reason, applicant submits that the Examiner failed to make a prima facie case for obviousness. Nevertheless claim 125 is a method claim while claim 1 is an apparatus claim. Therefore, applicant submits that the examiners reasoning given is not applicable for claim 125.

Claim 125 requires a method useful for editing an application that is built using components and that operates by generating documents. The first step is "*running the application, thereby executing selected components.*" As has been extensively explained in section A.ii and A.iii and with regard to claim 1, neither of the cited WebWriter references perform this step during editing – they do not run the application during editing. Applicant submits that for these reasons, claim 125 is patentable over the cited combination.

C. RESPONSE TO EXAMINER'S ARGUMENTS ON INDEPENDENT CLAIMS

i Argument (1)

On page 23 of the current action, the examiner states that the plain language of claims 1, 6 and 22 "*doesn't preclude, include or exclude when execution or running of the editor as taught by Web Writer is performed*." Applicant respectfully disagrees. In fact, claim 1 specifically requires that the document generator runs at least part of one of the applications and generates generated documents, the preamble says "applications generates generated documents" and that the generated documents include additional editing features. This makes it clear that the generated documents that result from running the application include editing features. In this way, editing and running can take place simultaneously.

In contrast, the WebWriter discloses two different programs that run at different times. The first program is the WebWriter editor, which includes its own page generation part that generates pages with editing features, but does not run the application. The second program is the WebWriter page generator, which is used only after editing, and which runs the application but does not insert any editing features. In contrast, claim 1 requires a single page generator that runs the application and, while doing so, inserts editing features into generated pages. Claim 22 in turn requires the editor program to operate on generated documents, whereby the generated documents result from a document generator executing the components. This also requires execution of components during editing because the editor operates on the output of the executing components.

Applicant notes that claim 125, for example, nicely recites running during editing by requiring a method for editing an application comprising the steps of running at least part of the application. Although the examiner's argument on page 23 is explicitly limited to claims 1, 6 and 22, the reasoning for claim 125 also refers back to the reasoning of claim 1.

The examiner further states that "Running saved content such as a template on the fly, in one or more steps isn't disclosed as not meeting Applicant's claimed limitations as recited in Applicant's claims." This statement appears to be a misinterpretation of applicant's argument. Applicant refers to p. 6, col. 2 of WebWriter I, in the section entitled "Running the Application:" "***Once*** *a Web Writer application is created and* ***saved*** *to disk it can be* ***run*** *in one or two ways*" (emphasis added). In applicant's reading, this sentence makes it crystal clear that the web writer

application runs **after** it is saved to disk, and the use of the word **once** at the beginning makes clear that the application does **not and cannot run before** it is saved to disk. In contrast, applicant's editor runs the application during editing, i.e., also before the application is saved to disk. As explained above, claim 1 requires running of the edited application during editing.

Further, on page 23 of the current office action, the examiner discusses "*features upon which applicant relies (i.e. not running only after editing and only running during editing)*" last full paragraph on page 23. In fact, applicant relies on the feature of running the edited web application during editing (. Applicant never stated that "only running during editing" is a distinctive feature. In fact, applications edited with applicant's editor keep running after editing as well. Applicant also never stated that "not running only after editing" is a distinctive feature. In fact, this phrase was introduced by the examiner and is very unclear. Applicant stated that "WebWriter runs the edited application only after editing", which means that WebWriter runs the application after editing but does not run the application during editing. In contrast, applicant's editor runs edited applications during editing and several claims require this as explained above.

The examiner seems to have taken applicant's statement that "WebWriter runs the edited application only after editing" and negated it into "not running only after editing" and then assumed applicant was relying on this as distinctive feature. In fact, the distinctive feature is best characterized as "running during editing." The recitation of this feature in the claims has been discussed above.

Since the examiner made incorrect assumptions about the distinctive features, his reasoning about these features in the claim language is not relevant.

The examiner further states that Web Writer does in fact disclose this limitation in his disclosure. Reading examiners reasoning "this limitation" seems to refer back to "only running during editing". Applicant strongly disagrees. Editors, including WebWriter and applicant's editor, produce applications that run after editing. An editor that runs the edited application only during editing would be grossly unusable, because it is the main task of an editor to produce applications that run at least after editing.

In the first paragraph of page 24, the examiner argues "it is actually quite inherent that the button is an executable component and hence this is being performed during editing", without actually giving a citation to support this. In fact, applicant believes that buttons of the

edited application in WebWriter are deactivated during editing and become working only after saving and running the application, because otherwise clicking a button would terminate the editor. Even more importantly, however, a button is just an HTML element and not a server side component or application. Claim 1, for example, explicitly says that it's a server side application running during editing, and Claim 6 recites server side components and "instructions contained in said selected component on the server".

The examiner then cites the 2nd paragraph of the WebWriter abstract: "WebWriter includes a direct manipulation Web page editor, the WebWriter Editor, which runs in the web browser as a CGI-service, and the WebWriter Page generator, which creates new pages as the application runs". The examiner then argues that the WebWriter page generator runs the application. Apparently, the examiner assumes that the WebWriter editor and WebWriter page generator work together during editing. This is, however, not the case. They are both separate programs and the WebWriter page generator is used only after the edited document is saved, as discussed above.

Finally the examiner states "in page 4 of Web Writer, under the Heading, "Creating the Template pages", Web Writer is said to be an editor that runs in a Web browser, ***which seamlessly generates/creates web pages while the application is run***." The underlined portion of this sentence is not actually included in the prior art reference, but has apparently been added by the examiner, and there is no support for such a statement.

ii Argument (2)

The examiner asserts that the plain language of claims 6 and 26 merely adds the limitation of having a similar appearance to the generated page with the addition of the editing features". Applicant notes that claim 6 refers to the similar appearance of a component, and adds a limitation "thereby calling first instructions" as well, while claim 26 also requires "functions similar".

Applicant reminds the Examiner that WYSIWYG editing was a significant break thru in editing technology by showing an editing view that is similar to the final end result. Therefore, applicant sees this as a very significant limitation rather than a minor limitation. It is a problem of WebWriter and many other editors of the time that this WYSIWYG principle is not followed for the dynamic portions of the pages.

The examiner then refers to the terms "edited template in WebWriter which contains both the static and dynamic portions" and "the created pages which are run in browser" and refers the 2nd paragraph on page 2 of the WebWriter I reference. Unfortunately, the cited portion neither contains these terms nor seems to discuss anything related. Apparently, the examiner believes both views appear similar in WebWriter I. This is not true, however, since the edited template in WebWriter which contains both the static and dynamic portions does not include the content of the dynamic portions but just the name of the script. Fortunately, the WebWriter article contains figures showing both views, which prove that they are different. Figure 7b (shown above) shows the "edited template in Web Writer which contains both the static and dynamic portions," and Figure 6b shows the "the created pages which are run in browser". Another pair of example figures is Fig. 11 and Fig. 10(b).

On page 25, the examiner argues that claim 6 and claim 26 are essentially the same, because the same extra limitation "of having a similar appearance to the generated page" has been added to the claims. This is not true because claim 6 has an extra limitation about the appearance of a *component* while claim 26 limits the appearance of a *page*. In addition, Claim 26 limits to appearance the function as well. Claim 6 has many limitations on the component, e.g. that component has to be server side, while claim 26 does not even mention the term component.

iii Claim 22

With respect to claim 22, the examiner states that applicant rehashes the same arguments as presented for claim 1, namely, that WebWriter does not execute part of the edited application during editing. Claim 22 differs from Claim 1 in that it explicitly requires editing and execution of components, while claim 1 does not refer to components at all. The examiner simply extended his argument regarding claim 1 and cites a button as a example of a component. Also, the argument on page 24 takes this same position, i.e., that a HTML button is a component running during editing. This argument related to claim 1 is simply not applicable to claim 22 since it does not take into account all the limitations on components expressly recited in claim 22, such as "a document generator program . . . for executing components" which rules out buttons as components, since HTML buttons are executed on client side by a web browser.

iv Claim 51 and claim 114

The Examiner maintains his rejection of these claims on the same grounds as claim 1, namely, on the basis that the Web Writer recites components such as buttons and contents computed on the fly. These features were also cited as teaching the limitations recited in applicant's claim 1. However, applicant submits that the citation of components in the context of claim 1 was superfluous and artificial, because claim 1 does not require components at all. The discussion on components in the context of claim 1 therefore cannot be transferred to any other claim, because claim 1 does not recite any limitations on components.

Further, Claims 51 and 114 introduce the limitations "features to cooperate in editing the component" and "features adapted to cooperate with an editor in editing said component," respectively, and these limitations have not been discussed before. The examiner's arguments regarding claim 1 are not applicable because of the following limitations on components within these claims. Claim 51 requires a plurality of components containing instructions to generate browser code and each component having features to cooperate in editing the component. In Response(1) on page 24 of the Action, the examiner gave a button as an example for a component. Since claim 51 requires components to generate browser code and buttons obviously do not, Response (1) is not applicable to claim 51. Also WebWriter's scripts are not components in the sense of claim 51, since they do not cooperate in editing the component because the WebWriter scripts are executed only at run time and not during editing, as discussed before.

Claim 114 in turn has limitations "components including first features adapted to cooperate with an editor in editing said component and second program instructions to generate browser code," which for the same reasons as claim 51 make the examiner's reasoning for claim 1 and Response(1) not applicable. Applicant also submits that claim 51 and claim 114 differ significantly, e.g., by claim 51 requiring "*a second software program transmitting … second documents to the first software program that make the first software program display a user interface for editing functions*".

v. Claim 59

With regard to claim 59, the examiner refers to Argument (1). However, applicant submits that claim 59 uses a very different limitation by requiring "the editor program

comprising first instructions for requesting that the document generator program processes a dynamic web document during editing". In fact, nothing in the reference teaches or suggests that the WebWriter editor requests that the WebWriter page generator generate a document. The WebWriter page generator is used at run time separately from the WebWriter editor.

vi. Claim 74

Claim 74 requires "the set of components on the generated documents can vary for different document requests for said document template."

The examiner maintains that this feature is taught by Web Writer I, stating that:

> "In Web Writer on page 2 of his disclosure, the editor cooperates with the browser as well as the page generator seamlessly allowing the user to interactively construct application without the need to learn HTML as disclosed in page 2, first paragraph of Web Writer I."

(See Action at p. 26) However, applicant disagrees and submits that, contrary to the Examiner's assertion, the cited portion does not disclose or suggest any cooperation between the WebWriter editor and the WebWriter page generator. If fact, there is no mention whatsoever of the page generator in the cited portion of the reference, but it does say: "To create template pages, the designer uses the WebWriter Editor . . . ." It also says the final steps of application building are "saving and loading a stack and running the application," but there is no indication at that point what causes the application to run. Moving forward to page 6 of the reference, it says that "Once a WebWriter application is created and saved to disk, it can be run in one of two ways," and here is where is becomes clear that the WebWriter Page Generator runs the application. Thus, the editor and the page generator have separate and distinct, clearly defined roles, and they do not cooperate to edit and run a web application. More importantly, however, the whole statement seems to be unrelated to the special limitation of claim 74 of a varying set of components. In Web Writer, the number of components per page is fixed as previously discussed.

vii Claim 90

With regard to claim 90, the examiner cites the following portion of WebWriter: "specifications of locations within the page, called *dynamic areas,* where computed content should be inserted **at runtime** to produce the final page that will be displayed to the user" (emphasis added). In applicant's reading, however, the cited portion makes it very clear that the

insertion is taking place at **runtime** and not at editing time. In addition, the portion refers to **computed** content which is inserted at runtime. It does not disclose **edited** content as argued by the examiner. In this context, the term "inserted" does not refer to an editing operation, but instead to an internal operation of the page generator taking place a runtime while executing the application. The result of the insertion is a temporary page being displayed in the browser, while an editing operation typically produces a document for storage as a template, not for temporary display. The cited portion therefore does not disclose editing at all as required by claim 90.

viii. Claim 125

With regard to claim 125, the examiner argues a similarity to claim 1. However, Claim 125 includes method steps that do not relate to any limitation recited in claim 1, for example, Claim 125 requires "executing selected components" while claim 1 does not mention components at all; Claim 125 requires "selecting a component by clicking" while claim 1 does not mention clicking; claim 125 requires "identifying the selected component in the source code of the application" while claim 1 does not recite identifying, components or source code. In addition, claim 125 nicely recites running during editing by reciting a method for editing a document having a step of running the application.

D. THE DEPENDENT CLAIMS ARE PATENTABLE OVER THE COMBINATION OF WEBWRITER I AND WEBWRITER II

1. Claims 2-5, 41 and 42

Claims 2-5, 41 and 42 are dependent from Claim 1 and should be considered patentable over the cited combination for all the same reasons.

With regard to claim 2, the Examiner cites page 2, column 1 of the WebWriter I article and refers to template pages, but does not explicitly describe what in WebWriter corresponds to the components. However, since the cited section introduces dynamic areas on the template pages, the examiner apparently interprets scripts called by dynamic areas of WebWriter as components. Claim 2 together with its base claim 1 requires a single page generator which is generating editing features and executing components. The WebWriter Page Generator. however, is only running at run-time, after editing, and executing scripts, while the WebWriter Editor generates editing features during editing and not at run-time. Since the WebWriter Editor

and the WebWriter Page Generator are two distinct programs running at different times, this is clearly distinct from a single page generator executing components and generating editing features at the same time. Therefore, applicant submits claim 2 is patentable over the cited combination.

Claims 3-5 are dependent on Claim 2, and for all the same reasons, applicant submits that claim 3-5 are not taught or suggested by the cited prior art.

With regard to claim 3, the examiner cited Fig. 2 of WebWriter I. However, the cited figure just shows the user interface of the editor. It does not show any dynamic areas and consequently does not give any information on components. In contrast, claim 3 requires "*at least one of the components that reacts interactively on user input by executing instructions of said component on the server*". As reasoned in section A.x, above, WebWriter does not teach or suggest anything like interactive components. For all these reasons, applicant submits that claim 3 is patentable over the cited combination.

Claims 4-5 are dependent on Claim 3, and for all the same reasons, applicant submits that claim 4-5 are not taught or suggested by the cited combination.

With regard to claim 4, the examiner cites WebWriter I Figs. 2 and 4. Fig. 4 shows a menu of available HTML tags, and does not give any information about dynamic areas. Fig. 2 also does not provide any information about dynamic areas. Claim 4, however, requires "*at least one of the components contains at least one other component*". Since components are apparently interpreted by the Patent Office as scripts associated with dynamic areas, both figures are then irrelevant for the showing one component containing another one. Section A.ix above explains that WebWriter does not teach or suggest one component containing another one. Therefore, claim 4 is patentable over the cited combination.

With regard to claim 5, the examiner cites page 9 of WebWriter I. However, page 9 reveals that each dynamic area is filled by its script exactly once per request. It is not possible that a script associated with a dynamic area is excluded from generation, or generated multiple times, and so the number of scripts of dynamic areas executed per template page is fixed. In contrast, the claim requires that the set of components on the generated documents can vary for different document requests. For all these reasons, applicant submits that claim 5 is patentable over the cited references.

With respect to claims 41 and 42, the examiner cites WebWriter II at page 1510, section 4.1, as disclosing a client part for execution on the client computer. Applicant refers to section A.v above for a discussion of the propriety of the combination of WebWriter I and WebWriter II. Specifically claim 41 read together with the base claim 1 implicitly requires the editor client part to work on the documents generated by the document generator program on the server ("an editor program dynamically operating on the generated documents"). Such a feature is not shown by WebWriter I, which works totally on the server side and does not have a client part. It is also not shown by WebWriter II, which includes parts that work on the client side. Thus, the combination of these different implementations is not readily obvious because it would require a significant effort to provide even just a basic compatibility between these two versions of WebWriter.

For all these reasons, applicant submits that claim 41 should be considered patentable over the cited combination. Claim 42 is dependent on Claim 41 and should be considered patentable over the cited combination for all the same reasons.

2. Claim 8

Claims 8 depends from Claim 6, and for all the same reasons is patentable over the cited combination.

With regard to claim 8, the examiner cited Fig. 4 of WebWriter I, just as with claim 4. Both claim 4 and claim 8 require a component containing another component. Therefore, applicant submits that claim 8 is patentable over the cited combination because of the reasons discussed with regard to claim 4. Applicant also refers to the reasoning in section A.ix above showing that WebWriter does not teach or suggest nested components. In contrast, claim 8 explicitly requires that a component is nested within a component

3. Claims 23-25

Claims 23-25 depend from Claim 22, and for all the same reasons, Claims 23-25 are not taught or suggested by WebWriter I, either alone or in combination with WebWriter II.

With regard to Claim 23, the examiner cited page 2 column 1 of WebWriter I and refers to use of the term "template page" as disclosing that the editor program operates functional applications in an edit mode. In applicant's claim, the word "applications" refers back to the base claim 22 and therefore means the applications being edited, not the editor itself. Claim 23

therefore requires the edited application be functional during editing by claiming "*the editor program operates functional applications in an edit mode permitting editing directly in the web browser*". Claim 23 requires that the application being edited is functional during editing by reciting "operates a functional application" and "editing said application". As reasoned in section A.i above, page 2, column 1 of the WebWriter I article does not state that the edited application is running during editing ("*the static parts of each page are created in advance by the designer using a text editor*"). Section A.ii and A.iii gives several citations to the WebWriter articles that support the premise that, in fact, the application is run only after editing and not during editing. During editing, dynamic parts are displayed as placeholders (See page 4, col. 2, last para.: "*WebWriter then adds a placeholder for the output area*"). For all these reasons, applicant submits that WebWriter does not teach or suggest "the editor program operates functional applications" as required by claim 23 and therefore, claim 23 is patentable over the cited combination.

Claims 24-25 depend from Claim 23, and for all the same reasons are patentable over the cited combination.

With regard to Claim 24, the examiner cited pp. 2-9 of WebWriter I as disclosing a component that can react on subsequent document requests by executing selected instructions. Applicant submits that the scripts called by WebWriter's dynamic areas are just used for processing of a single request. As explained on page 9, left column, heading "The Web Writer Page Generator," the scripts are called during document generation to produce output for an output area during processing of a single request. The scripts do not have a concept of multiple requests or subsequent requests and can therefore also not react in any way on a subsequent request. In contrast, claim 24 requires at least one component that can react on subsequent document requests. Please also refer to section A.x above. For all these reasons, applicant submits that claim 24 is patentable over the cited combination.

Claim 25 depends from Claim 24, and for all the same reasons is patentable over the cited combination.

For Claim 25, the examiner cited WebWriter I pp. 2-9, et seq. as disclosing component classes. However, applicant submits that neither WebWriter I nor WebWriter II discloses component classes implementing components. The scripts associated with dynamic areas are Unix scripts or programs as disclosed on WebWriter I page 9, second column first paragraph

("*runs the named program … The program is run in a separate Unix process*"). So the WebWriter Page Generator in fact calls Unix scripts or programs. There is no teaching or suggestion in WebWriter that components are objects and implemented as classes.

In addition, WebWriter does not have a menu of components for insertion into the document templates. Figure 7(b) of the WebWriter I article shows forms for entering scripts, but not a menu.

Applicant submits that claim 25 is patentable over WebWriter I alone or in combination with WebWriter II since WebWriter does not teach or suggest component classes, a document generator program using component classes, nor a menu of components.

4. Claims 27-33, 43

Claims 27-33 and 43 depend from Claim 26, and for all the same reasons are patentable over the cited combination.

With regard to dependent claim 27, the examiner cites WebWriter I pages 2-9 et seq. as disclosing at least one component being executed by the first software program. Since base claim 26 requires the first software program to generate a second document having features which permit editing, the examiner must be interpreting the WebWriter Editor as the first software program. According to WebWriter I page 9 section entitled "The WebWriter Page Generator" (see also section A.iii above), it is the WebWriter Page Generator executing scripts associated with dynamic areas (see also Section A.vii) and not the WebWriter Editor. In contrast, the claim requires the first software program to execute at least one component. Therefore, applicant submits that claim 27 is patentable over the cited combination.

With regard to claim 28, the examiner cites WebWriter I page 4 for "clicking on its handles" as disclosing that the second document includes handles and choosing one of the handles selects a component for an editing operation. According to the limitations of base claim 28, the second document is a document that already contains code generated by executing the components. So, the claim language indeed requires handles to occur in a document generated by executing components. In contrast, WebWriter shows handles during editing inside a document that contains placeholders. In addition, the cited portion of the WebWriter I article seems to deal with HTML elements rather than dynamic areas, which are discussed later in the article. For all these reasons, applicant submits that claim 28 is patentable over the cited combination.

Claim 29 depends from Claim 28, and for all the same reasons is patentable over the cited combination. With regard to claim 29, applicant submits that the delete operation is not described in the cited portion of the WebWriter I reference.

With regard to dependent claim 30, the examiner cites WebWriter I pages 2-9 et. seq. as disclosing that the features include scripts. However, applicant disagrees. The WebWriter Editor seems to send HTML documents, which do not contain scripts. In contrast, the claim implicitly requires the second document to contain scripts, since claim 30 requires the features to include scripts and base claim 26 requires the second document to include these features. Therefore, applicant submits that claim 30 is patentable over the cited combination.

Claim 31 depends from Claim 30, and for all the same reasons is patentable over the cited combination. With regard to dependent claim 31, the examiner cites WebWriter I pages 2-9 et. seq. as disclosing scripts that encapsulate information from the first document. As discussed with regard to claim 30, the WebWriter Editor does not work to include scripts into the generated documents. In contrast, the claim explicitly requires scripts that encapsulate information from the first document, and therefore applicant submits that claim 31 is patentable over the cited combination.

With regard to dependent claim 33, the examiner cites WebWriter II, page 1510 section 4.1 Applicant submits that a combination of WebWriter II and WebWriter I is improper as reasoned in section A.v above, and for that reason, the citation of WebWriter II for the dependent claim is inadequate to teach or suggest the claimed invention.

In addition, the cited portion does not seem to reveal change requests. The WebWriter II seems to send a complete document to the server after editing on a save operation. On page 1511, 3rd paragraph: "The CGI modules provide server-side services such as loading files, parsing HTML, **saving files** …" (emphasis added). In contrast to saving a complete document, claim 33 requires "*to send change requests for the first document to the server computer*", which mandates that a change be sent to the first document and not a complete document. For these reasons, applicant submits that claim 33 is patentable over the cited combination.

With regard to dependent claim 43, the examiner cites WebWriter II, page 1510 section 4.1. Applicant submits that a combination of WebWriter II and WebWriter I is improper as reasoned in section A.v, and for that reason, this claim is considered patentable over the cited combination.

5. Claims 52-58

Claims 52-58 depend from Claim 51, and for all the same reasons, these claims are patentable over the cited combination.

With respect to claim 53, the examiner cited WebWriter II page 1508 section 1.2, while he cited WebWriter I for the base claim 52 and 51. Applicant submits that for all the reasons given in section A.v the combination of WebWriter I and WebWriter II is improper, and therefore the obviousness rejection should be withdrawn.

With respect to claim 54, the examiner cites WebWriter I pages 2-9 et. seq. as disclosing second documents that include HTML pages with embedded scripts. However, in the WebWriter I article, the WebWriter Editor does not use client side scripts. In contrast, the claim requires second documents to contain embedded scripts, and base claim 51 requires the second program to generate the second documents, and therefore applicant submits that claim 54 is patentable over the cited combination.

With regard to claim 55, the examiner cites WebWriter I pages 2-9 and refers to use of the term "editor" as disclosing edit functions include removing a component. The WebWriter I article does not appear to describe the function of deleting a component.

With respect to claim 57, the examiner cites WebWriter pages 2-9 and refers to the term "editor" as disclosing that the generated documents include edit features. The term generated document, however, refers back to claim 56, where it is used as the documents generated by the WebWriter Page Generator and by executing instructions of the components. As reasoned in section A.iii, however, the WebWriter Page Generator is separate from the WebWriter Editor and does not insert any editing features into the generated documents. In contrast, the claim specifically requires generated documents to include editing features. Therefore, applicant submits that claim 57 is patentable over the cited combination.

With respect to claim 58, the examiner cites WebWriter I pages 2-9 and refers to the term "editor" as disclosing instructions to allow the user to click on the generated documents to select items to perform edit functions on. The term generated documents, however, refers back to claim 56 , where it is used as the document generated by the WebWriter Page Generator and by executing instructions of the components. As reasoned in section A.iii, however, the WebWriter Page Generator is separate from the WebWriter Editor. Therefore it is not possible to click on a

document generated by the WebWriter Page Generator to perform editing. In contrast, the claim requires instructions to allow the user to click on the generated document to select items to perform edit functions on. Therefore, applicant submits that claim 58 is patentable over the cited combination.

6. Claims 60-73

Claims 60-73 are dependent from Claim 59, and for all the same reasons, applicant submits that Claims 60-73 are likewise not taught or suggested by WebWriter I, either alone or in combination with WebWriter II.

With regard to claim 60, the examiner cites section 4 of the WebWriter II article, while citing the WebWriter I article for base claim 59. Applicant submits that for all the reasons given in section A.v above, the client side operations described in section 4 of WebWriter II cannot easily be combined with the server side functionality of WebWriter I. Therefore, the combination is improper and not effective to make the claims obvious.

Claims 61-63 depend from Claim 60, and for all the same reasons, these claims are patentable over the cited combination.

With regard to claim 61, the examiner cited WebWriter I page 2-9 as disclosing further instructions for execution during document generation to collect edit information for use by the editor. In fact, the WebWriter I article describes two kinds of document generation, the document generation done by the WebWriter Editor on page 8, and the document generation done by the WebWriter Page Generator on page 9. According to claim 59, "document generation" means "generating generated documents from dynamic web documents which look and function similar to end user's view of the documents". This is not the case for the document generation of the WebWriter Editor (see section A.ii). For the document generation of the WebWriter Page Generator, there is no teaching or suggestion of a feature "to collect edit-information" as required by the claim. Therefore, applicant submits that claim 61 is patentable over the cited combination.

With regard to Claim 63, the examiner cites WebWriter I page 2-9 for automatically repeating requesting that the document generator process the dynamic web document if required. As reasoned with regard to claim 59 and explained in section A.ii and A.iii, the WebWriter Editor never does request that the WebWriter Page Generator generate a document, and so there

is also not a repeating of this step in WebWriter I. In fact, the WebWriter Page Generator is only running at run-time when the WebWriter Editor has already saved the document. Therefore, applicant submits that claim 63 is patentable over the cited combination.

With regard to Claim 66, the examiner cites page 2-9 of the WebWriter I article as disclosing the operations of inserting, deleting and modifying a component. However, the WebWriter I article does not describe the deletion of a component.

With regards to Claim 67, the examiner cites page 2-9 of WebWriter I as disclosing that the view looks, except for editing features, similar to the end-user view. In general, both views do not look similar as reasoned in section A.ii above. In fact, Figs. 7(b) and 6(b) of WebWriter I show both views, which clearly look different since the end user's view contains the output generated by scripts associated with dynamic areas while the view shown by the editor just shows placeholders. Also, see Figs. 10(b) and 11 for another example. In contrast, the claim requires both views to look similar except for editing features. Therefore, applicant submits that claim 67 is patentable over the cited combination.

With regard to claim 68, applicant refers to the reasoning for claim 61.

Claim 69 is dependent on 68, and for all the same reasons, applicant submits that Claim 69 is likewise not taught or suggested by WebWriter I, either alone or in combination with WebWriter II. With regard to claim 69, the examiner cited WebWriter I at pp. 2-9 as disclosing the editor using the edit information to modify the dynamic web document. However, in WebWriter I, the edit information is never collected as explained above with respect to claim 61. Therefore, applicant submits that claim 69 is patentable over the cited combination.

Claim 70 is dependent on 69, and for all the same reasons, applicant submits that Claim 70 is likewise not taught or suggested by WebWriter I, either alone or in combination with WebWriter II. With regard to claim 70, the examiner cites WebWriter I at pp. page 2-9. Claim 70 requires "*position information on the components contained in the document template*". Applicant submits that WebWriter I does not contain any teaching or suggestion of such position information. Applicant therefore submits that claim 70 is patentable over the cited combination.

With regard to claim 72, the examiner cited WebWriter I at pp. 2-9 as disclosing initiating a reload in the browser. A reload in the browser means a function of the browser to load the same page a **second time**. However, WebWriter I does not describe the editor as

containing instructions for initiating a reload, as required in Claim 72. Therefore, applicant submits that claim 72 is patentable over the cited combination.

With regard to claim 73, the examiner cites WebWriter I at pp. 2-9. In contrast, claim 73 requires information on an element to be displayed **without** requesting the document generator. In contrast, WebWriter I **does** have page generation in the WebWriter Editor when showing information on a dynamic area and associated script (as reasoned below). In that situation, there is no page generation in the WebWriter Page Generator since the WebWriter Page Generator runs only at run-time after editing. However, as explained in the reasoning of claim 59 and section A.ii, the WebWriter Page Generator is not a page generator in the sense of base claim 59 and therefore not relevant for the claim.

In order to display information on an element the user must click a handle. The 2nd paragraph on page 9 left column states: "The next time WebWriter is called (after the user clicks on a handle or control panel button)," which makes clear that clicking a handle makes WebWriter request the WebWriter Editor to generate a document. Applicant's editor in contrast does have a client part, that can process selected mouse clicks on handles without requesting that the server part generate a page (and consequently without delay of a server interaction). So, because WebWriter I requires a page generation to display information on an element but claim 73 requires this operation to be done without performing a request, the claim is patentable over WebWriter I.

7. Claims 75-89

Claims 75-89 depend from claim 74, and for all the same reasons, should be considered patentable over the cited combination.

With regard to claim 75, the examiner cites WebWriter I at pp. 2-9 and refers use of to the term "editor" as disclosing editing functions comprising adding, modifying and deleting a component. However, the function of deleting a component does not seem to be described in the cited document.

With regard to claim 76, the examiner cited WebWriter I at pp. 2-9 and refers to use of the term "templates" as disclosing components denoted on document templates using tag syntax, whereby the tag name identifies the component kind. However, WebWriter I on page 9 left column discloses that output areas with scripts are denoted using a tag of the form:

<output script="…">formatting string</output>.

The tag name of this syntax is "output" while it is the script attribute that identifies what the script is to be called. In contrast, however, claim 76 explicitly requires the tag name to identify the component kind. Therefore, applicant submits that claim 76 is patentable over the cited combination.

With regards to claim 77, the examiner refers to a combination of WebWriter I and WebWriter II. Applicant refers to section A.v and submits that a combination of WebWriter I and WebWriter II is improper, and should be withdrawn as a basis for rejection.

With regard to claim 78, the examiner cites WebWriter I at pp. 2-9 and refers to use of the terms "generator" and "templates" as disclosing that at least one component can be excluded from said generated document. However, the section "The Web Writer Page Generator" on page 9 makes clear that the WebWriter Page Generator processes each dynamic area on a template page by executing the associated script and has no ability to exclude a dynamic area from this process. In contrast, the claim requires "*at least one component that … can be excluded from said generated document upon selected document requests for said document template*".

In addition, the claim requires a component that can react interactively on subsequent document requests. As discussed in section A.x, WebWriter does not seem to disclose interactive components. For all these reasons, applicant submits that claim 78 is patentable over the cited combination.

Claims 79-80 depend from Claim 78, and for all the same reasons, these claims are patentable over the cited combination.With regard to claim 79, the examiner cites WebWriter I at pages 2-9 as disclosing instructions to prevent excluded components from reacting on subsequent document requests. As discussed in section A.x, scripts of dynamic areas of WebWriter cannot react on subsequent document requests and therefore WebWriter also does not include instructions to prevent this reacting.

Claim 80 depends from Claim 79, and for all the same reasons, this claim is patentable over the cited combination.With regard to claim 80, the examiner cites WebWriter I at pages 2-9 as disclosing storing information in session memory on some components that are present on the document generated. However, the section "The Web Writer Page Generator" on page 9 does not refer to session memory or storing of information in session memory on scripts of dynamic areas. The claim refers to actions taking place at run-time and therefore would have been described in

the cited section if they had been present in WebWriter I. Thus, applicant submits that there is no teaching or suggestion for storing of information in session memory on scripts of dynamic areas in WebWriter and therefore the claim should be patentable over WebWriter.

With regard to claim 81, the examiner cites WebWriter I at pages 2-9 and refers to use of the term "template" as disclosing that at least one component can be excluded from said generated document. However, the section "The Web Writer Page Generator" on page 9 makes clear that the WebWriter Page Generator processes each dynamic area on a template page by executing the associated script and has no ability to exclude a script of a dynamic area from this process. In contrast, the claim requires "*instructions to decide about exclusions of components*".

In addition, claim 81 requires components nested inside the first component. Applicant refers to section A.ix for the discussion that WebWriter does not disclose nested components. For all these reasons, applicant submits that claim 81 is patentable over the cited combination.

With regard to claim 82, the examiner cites WebWriter I at pages 2-9 and refers to the term "display/interface" as disclosing an editor taking the varying set of components into account. However, the WebWriter Editor displays each dynamic area with associated script as exactly one placeholder. There is no mechanism in the WebWriter Editor to take a varying set of components into account. In contrast, the claim requires the editor be able to take the varying set of components into account. Also, as discussed with regard to claim 74, the WebWriter Page Generator cannot handle a varying set of components. For all these reasons, applicant submits that claim 82 is patentable over the cited combination.

With regard to claim 83, the examiner cited WebWriter I at pages 2-9 and refers to the terms "editor" and "template" as disclosing an editable view that includes and excludes selected components. However, the WebWriter Editor displays each dynamic area with associated script as exactly one placeholder. There does not seem to be a view that includes and excludes placeholders. In contrast, claim 83 requires a view that includes and excludes selected components. For all these reasons, applicant submits that claim 83 is patentable over the cited combination.

With regard to claim 84, the examiner cites WebWriter I at pages 2-9 and refers to the term "template" as disclosing a document generated for at least one document template that contains more components than the document template. In WebWriter, each dynamic area is filled by its script exactly once, as explained in section "The Web Writer Page Generator" on

page 9. Therefore, a page generated by the WebWriter Page Generator cannot contain more components than the template page. In contrast, however, the claim requires that a generated document contains more components than the document template. Therefore, applicant submits that claim 84 is patentable over the cited combination.

With regard to claim 85, the examiner cites WebWriter I at pages 2-9 and refers to use of the term "template" as disclosing multiple instances of a component being included in a generated document. In WebWriter, each dynamic area is filled by its script exactly once, as explained in section "The Web Writer Page Generator" on page 9. In contrast, claim 85 requires multiple instances. Therefore, applicant submits that claim 85 is patentable over the cited combination.

With regard to claim 86, the examiner cited WebWriter I at pages 2-9 as disclosing instructions to assign unique identifiers and to qualify names generated into the browser code with the unique identifiers. The cited portion, however, does not seem to reveal assigning unique identifiers or qualifying names. In particular, the section "The Web Writer Page Generator" on page 9, which describes the run-time page generation in the WebWriter Page Generator, does not describe this feature. Thus, applicant submits that claim 86 is patentable over the cited combination because the cited art does not seem to teach or suggest assigning of unique identifiers and qualifying names as required by the claim.

With regard to claim 87, the examiner cites WebWriter I at pages 2-9 and refers to use of the term "template" as disclosing instructions to decide how many instances of components are included in the documents generated. As reasoned with regard to claim 85, WebWriter does not teach or suggest multiple instances and therefore it is clear that it also does not have instructions to decide how many instances of components are included in the documents generated. In contrast, instructions to decide how many instances of components are included in the documents generated are specifically required by the claim. Therefore, applicant submits that claim 87 is patentable over the cited combination.

With regard to claim 88, the examiner cites WebWriter I at pages 2-9 and refers to use of the term "template" as disclosing an editable view that includes multiple instances. As reasoned with regard to claim 85, WebWriter does not teach or suggest multiple instances at run-time and therefore it is clear that does not have multiple instances in an editable view. Also, the description of the WebWriter Editor does not reveal multiple instances of placeholders or of

scripts of dynamic areas. In contrast, the claim explicitly requires an editable view that includes multiple instances of selected components. Therefore, applicant submits that claim 88 is patentable over the cited combination.

With regard to claim 89, the examiner cites WebWriter I at pp. 2-9 and refers to use of the term "browser" as disclosing a sixth component including tenth instructions for displaying the sixth component during editing. Applicant submits that as reasoned in section A.ii the WebWriter Editor shows placeholders during editing and does not execute any instructions of the scripts associated with dynamic areas during editing. Therefore, applicant submits that WebWriter does not disclose tenth instructions. In contrast, the claim requires a sixth component including tenth instructions for displaying the sixth component during editing. Therefore, applicant submits that claim 89 is patentable over the cited combination.

8. Claims 91-113

Claims 91-96 depend from Claim 90, and for all the same reasons are patentable over the cited combination.

With regard to claim 91, the examiner cited WebWriter I at pages 2-9 as disclosing a second window. In applicants reading, WebWriter I just uses a single window, for example, as shown in figure 1. In contrast, claim 91 explicitly requires a "second window". WebWriter II appears to use frames and not a second window. For all these reasons, applicant submits that claim 91 is patentable over the cited combination.

With regard to Claim 93, the examiner cites WebWriter I at pages 2-9 as disclosing that the original document looks similar to the document. Applicant submits that in general the original and the document shown during editing look clearly different in WebWriter since dynamic areas are replaced by placeholders. This is discussed in section A.ii. Therefore applicant submits that claim 93 is patentable over the cited combination.

Claims 94-95 depend from Claim 93, and for all the same reasons are patentable over the cited combination. With regard to Claim 94, the examiner cites WebWriter I at pages 2-9. The claim explicitly requires the original document to be a dynamic document containing components with instructions contained therein. As explained in section A.ii, in the case of dynamic documents, the WebWriter Editor displays a document that appears different than the original document since dynamic areas are replaced by placeholders. In contrast, base claim 93

requires that the original document and the document look similar. Therefore, applicant submits that claim 94 is patentable over the cited combination.

With regard to claim 96, the examiner cites WebWriter I at pages 2-9 as disclosing that links contained in the document stay functional during editing. The WebWriter I article does not include any teaching on how links are displayed in a document being edited. Applicant further submits that simply including a link in a document edited by the WebWriter Editor would leave the editor and navigate to the linked page. In contrast, applicant's claim requires that links stay functional during editing in order to allow the user to browse and edit at the same time. For all these reasons, applicant submits that claim 96 is patentable over the cited combination.

9. Claims 115-124

Claims 115-124 and 128 depend from claim 114, and for all the same reasons, should be considered patentable over the cited combination.

With regard to claim 115, the examiner cited WebWriter I at pages 2-9 and refers to use of the term "editor" as disclosing that first features include instructions for passing information to the editor. According to base claim 114, components for execution on the server include first features for cooperation with the editor. Claim 115 states "*first features include fourth program instructions for passing information to the editor*" and depends on claim 114, which states that "*at least one of the components including first features.*" WebWriter does not describe that scripts called by WebWriters dynamic areas pass information to the editor. In fact, as discussed in sections A.vii, A.ii and A.iii, these scripts are executed at run-time, after editing is finished, and so the scripts running later cannot pass information to the editor as required by the claim. Consequently, WebWriters scripts called by dynamic areas have no features for passing information to the editor. In contrast, claim 115 requires these features and therefore applicant submits that claim 115 should be patentable over the cited combination.

Claims 116-118 depend from Claim 115, and for all the same reasons, these claims are patentable over the cited combination.

With regard to claim 116, the examiner cited WebWriter I at pages 2-9 and refers to use of the term "editor" as disclosing that part of the information is collected during execution of the components on the server. However, the scripts called by WebWriter's dynamic areas do not collect information for the editor. As discussed in sections A.vii, A.ii, and A.iii, these scripts run

at run-time after editing is finished and therefore cannot pass information to the editor. In contrast, claim 116 requires that part of the information is collected during execution of the components. Therefore, applicant submits that claim 116 should be patentable over the cited combination.

With regard to claim 117, the examiner refers to WebWriter II. Applicant submits, however, that "said information" is in fact never collected as reasoned with regard to base claim 115 and therefore also not transmitted either by WebWriter I or by WebWriter II.

With regard to claim 118, the examiner cited WebWriter I at pages 2-9 and refers to use of the term "editor" as disclosing that the information includes attributes of the component. As reasoned with regard to base claims 114 and 115, no information is transmitted from the running scripts associated with WebWriter's dynamic areas to the WebWriter Editor. In addition, the cited portion does not show that the scripts called by WebWriter's dynamic areas collect any attribute values to be passed to the editor. In contrast, base claims 114 and 115 require components to include fourth program instruction for passing information to the editor, and claim 118 requires the information to include attribute values of components. For these reasons, applicant submits that claim 118 is patentable over the cited combination.

With regard to claim 119, the examiner cited WebWriter I at pages 2-9 and refers to the use of term "editor" as disclosing first features including fifth instructions that display additional editing features. The base claim 114 requires "*components including first features*". In applicants reading, the cited portion does not reveal that scripts called by WebWriter's dynamic areas display any additional editing features. In fact, as reasoned in sections A.vii, A.ii, and A.iii, these scripts run at run-time only, i.e., after editing. Therefore, it makes no sense for them to display any editing features since editing is already finished. In contrast, claim 119 together with base claim 114 requires components including first features including instructions that display additional editing features. For these reasons, applicant submits that claim 119 is patentable over the cited combination.

Claim 120 depends from Claim 119, and for all the same reasons, this claim is patentable over the cited combination. With regard to claim 120, the examiner cites WebWriter I at pages 2-9 and refers to use of the term editor as disclosing that said editing features include handles. In applicants reading, handles are displayed by the WebWriter Editor. In contrast, claim 120 requires said editing features to include handles, base claim 119 requires first features include

instructions that display additional editing features, and base claim 114 requires components including first features. So, the claim requires **components** to contain instructions to display handles. In contrast, in WebWriter it is the editor displaying the handles. For these reasons, applicant submits that claim 119 is patentable over the cited combination.

With regard to claim 121, the examiner cited WebWriter I at pages 2-9 and refers to the term editor as disclosing first features including extensions for use by the editor. Claim 121 recites "*first features include an extension for use by the editor"* and read together with the base claim 114 "*at least one of the components including first features*" limits the extensions to be part of the components and not of the editor. The cited portion of WebWriter does not reveal any extensions for use by the editor being contained in the scripts called by WebWriter's dynamic areas. The scripts are used only at run time, when the editing is finished, as reasoned within sections A.vii, A.ii and A.iii. Therefore, applicant submits that WebWriter does not teach extensions as required by the claim. For these reasons, applicant submits that claim 121 is patentable over the cited combination.

Claim 122 depends from Claim 121, and for all the same reasons, this claim is patentable over the cited combination. With regard to claim 122, the examiner cited WebWriter I at pages 2-9 and refers to use of the term "editor" as disclosing a web page for editing component attributes. Claim 122 explicitly requires "*said extension enables display of a page for editing the components attributes values.*" As reasoned with regard to base claim 121, the base claims require the extension to be part of components. In contrast, all the forms shown by the WebWriter Editor are built into the editor itself. For these reasons, applicant submits that claim 122 is patentable over the cited combination.

With regard to claim 123, the examiner cited WebWriter I at pages 2-9 and refers to use of the term "templates" as disclosing components denoted on document templates using tag syntax, whereby the tag name identifies the component kind. However, WebWriter I on page 9 left column discloses that output areas with scripts are denoted using a tag of the form

<output script="…">formatting string</output>

The tag name of this syntax is "output" while it is the script attribute that identifies what the script is to be called. In contrast, however, claim 123 explicitly requires the tag name to identify the component kind. Therefore applicant submits that claim 123 is patentable over the cited combination.

With regard to claim 124, the examiner cited WebWriter I at pages 2-9 and refers to use of the term "browser" as disclosing second program instructions to generate browser code for displaying the component during editing. According to the base claim 114, second program instructions are part of selected components. As reasoned within sections A.vii, A.ii, and A.iii, WebWriter's scripts associated with dynamic areas are not executed during editing. The WebWriter Editor just displays a placeholder for scripts of dynamic areas during editing. In contrast, claim 124 requires second instructions in the components for displaying the components during editing. Therefore, applicant submits that claim 124 is patentable over the cited combination.

10. Claims 126-127

Claims 126-127 depend from Claim 125, and for all the same reasons are patentable over the cited combination.

With regard to claim 126, the examiner cited WebWriter I at pages 2-9 as disclosing repeating the running and the displaying steps after applying a modification function. In applicants reading, as reasoned in sections A.ii and A.iii, WebWriter runs the application only after editing is finished and the document is saved. So there is no running step during editing and consequently also no repeating of the running step in WebWriter. In contrast, the claim requires repeating the running and the displaying steps after applying a modification function. Therefore, applicant submits that claim 126 is patentable over the cited combination.

Claims 127 depends on Claim 126, and for all the same reasons is patentable over the cited combination.

E. RESPONSE TO EXAMINERS ARGUMENTS ON DEPENDENT CLAIMS

With regard to claim 2-5, the examiner alleges that buttons are components in the sense of the claim. Applicant disagrees since claim 2 requires components which are executed by the document generator, using the claim language "wherein the document generator executes selected components on document templates." However, an HTML button is a client side part of the browser.

With respect to claims 4 and 5, the examiner takes the fact that a page template contains a dynamic area as an example of one component containing another component. However, the

base claim 2 already makes it clear that document templates contain components (“document template capable of containing components”) and so only items that can be placed on page templates are components in the sense of base claim 2; and this is not the case for document templates. So a page template containing a component is not an example for nested components. Claim 4, however, explicitly recites that one component contains another component (see Section A.ix).

With respect to claims 41 and 42, the examiner argues that applicant is simply rehashing previous arguments as discussed above in response to Argument (1). Nevertheless, claims 41 and 42 introduce the client part into the claims, and the examiner recites page 1510 of the WebWriter II, which is not cited for claim 1 and not used in Response(1).

With regard to claim 23, the examiner refers to his argument for claim 90. Applicant asserts that, as described above, the examiner’s argument only refers to run-time and does not give information on editing-time and is therefore not applicable to claim 23.

With respect to claim 24, the examiner states that the plain language of the claims discloses reacting to "subsequent document requests" and not to multiple requests. However, applicant’s argument was that WebWriter’s scripts work only during a single request and not for more than one or multiple requests. A component able to handle subsequent requests, however, must be able to distinguish between an initial and a subsequent request. Under the Heading “The Web Writer Page Generator,“ the WebWriter I reference only describes the initial request that displays a dynamic area. It does not give any information on any subsequent requests, i.e. requests following the initial one.

With respect to claim 25, the examiner argues applicant is rehashing previous arguments. However, claim 25 claims new features, a component class and a menu of components, which are not recited in any of the claims discussed previously.

With respect to claims 27-33 and 43, the examiner argues that applicant is rehashing previous arguments. However, claim 28 introduces the term “handles,” claim 29 adds the delete operation, claim 30 recites “features including scripts,” whereby “features permit editing of the first document,” claim 31 adds “scripts that encapsulate information from the first document,” claim 33 includes “change requests.” All these features have not been recited and discussed in other claims before.

The examiner then reasons about handles and states "Examiner again disagrees, WebWriter on page 2 of his disclosure, shows dynamic areas within the template which is edited and also executed." Since the examiner does not give a more detailed reference, applicant assumes that the examiner refers to portions of page 2 that have already been discussed. In applicant's reading of page 2, dynamic areas are executed after editing and not during editing, e.g. "*dynamic areas, where computed content should be inserted at* ***runtime***", page 2 left column last full paragraph, emphasis added.

With respect to claim 52-58 the examiner argues applicant is rehashing previous arguments. However, the base claim 51 adds specific limitations on components "*having features to cooperate in editing the component*" which are important for the dependent claims. In addition claim 52 introduces "browser code that can differ for multiple requests", claim 54 adds "program instructions for collecting and passing information to the editor", claim 55 includes various "editing functions", claim 56 details the use of fourth instructions, claim 58 adds to "click on the generated documents to select items". All these features have not been recited and discussed in other claims before.

With respect to claims 60-69, the examiner argues applicant is rehashing previous arguments. Applicant submits, however, that claim 61 introduces "to collect edit information for use by the editor"; claim 63 adds "automatically repeating the request", claim 69 includes "using the edit information to modify the dynamic web document". All these features have not been recited and discussed in other claims before.

The examiner then asserts "Applicant argues that the web generator as taught by Web Writer doesn't disclose generating documents from dynamic web documents." This is not a correct characterization of applicant's argument. In fact, there is no program in the WebWriter article that is named "web generator" and applicant's argument also does not use that term. As already discussed in detail, the WebWriter article discloses two programs for generating pages. In fact, Applicant argues that the WebWriter Page Generator does not collect edit information while creating new pages. In contrast, claim requires fourth instructions for execution during the document generation to collect edit-information.

With respect to claim 70, applicant argues that Web Writer in combination with Web Writer II doesn't disclose position information regarding how components are placed on the template. The examiner disagrees and recites under the heading Web Writer editor, that "... *Web*

Writer inserts a new object of *the requested class into the content tree data structure at the position designated by the insertion point and sets this new object to be the selected object... ".* In applicant's reading, this discloses position information in the document tree and not position information on the document template. Claim 70, however, requires position information on selected components marked on the dynamic web document.

With respect to claim 72, applicant argues that the examiner did not give a prior art reference for the editor having instructions "for initiating a reload in the browser". The examiner now recites a new portion on page 8 just on top of the headline "Discussion of the WebWriter Approach" reciting refresh meta tags. The cited portion, however, discusses the Walkie-talkie sample application and proposes to add the refresh meta tag to that application, and not to the editor itself. Claim 72, however, requires the reload to be performed by first instructions, which are part of the editor as claimed in the base claim 59. In addition, the refresh meta tag technically does a redirect to another page rather than reloading the current one. Even if the meta tag is used to redirect to the same page, such an operation loses possible URL or form parameters. In contrast, claim 72 explicitly requires a reload in the browser. The examiner also cites an operation of loading a stack. This operation is clearly different from the well defined reload function of a web browser.

With respect to claim 73, the examiner writes "applicant argues that there is no page generation in the web generator taught by Web Writer." This sentence has been taken out of context as it was used in to discuss an editing situation, and while editing, indeed, the WebWriter page generator is not used. Please note that claim 73 uses very special negative claim language "to display information on at least one element"… "without requesting that the document generator program generates a document" which is not used in other claims. Applicant clarified his reasoning on claim 73.

With regard to claim 75-89, the examiner writes applicant similarly rehashes arguments as previously addressed above. Applicant submits, however, that Claim 76 introduces tag syntax and tag name, Claim 78 introduces excluding of components, Claim 79 introduces reacting of excluded components, Claim 80 introduces storing information on selected components in session memory, Claim 81 introduces deciding exclusion of nested components, Claim 82 introduces an editable view with varying set of components, Claim 83 introduces an editable view that includes and excludes components, Claim 84 introduces more components on page that

on the template, Claim 85 introduces multiple instance of components, Claim 86 introduces qualifying of names, Claim 87 introduces deciding how many components, Claim 88 introduces an editable view with multiple components, Claim 89 introduces tenth instructions to display component during editing and at runtime. All these features have not been recited and discussed in other claims before. In addition, the base claim 74 introduces a new feature of a varying set of components per page, which is not recited in any of the other dependent claims.

With respect to claim 77, the examiner argues "Web Writer II essentially offers an improvement over the initial Web Writer version". Applicant strongly disagrees. Although one might assume from the naming of WebWriter and WebWriter II, WebWriter II is not a newer improved version of WebWriter. WebWriter II is a totally different approach for constructing an editor. WebWriter II has an implementation architecture (called "Meteor shower") which is totally different and incompatible to WebWriter I. It also has other advantages and disadvantages than WebWriter I. While WebWriter is essentially a C++ program running on a server, WebWriter II is a javascript program running on a client. As reference *see* WebWriter II, page 1508, second paragraph. Moreover, the new Meteor Show architecture teaches away from running during editing, since the editor essentially operates on the client. In contrast, execution of a server side web application (during editing or not) is server side.

With respect to claim 80, the examiner introduces related work for "saving and restoring the state of runtime data structures." Claim 80 however contains limitations to store specific information in session memory "store information in session memory on some of the components that are present on the document generated" and also limitations on how that information is used later on "only react on components that have been remembered in session memory". The recited portion does not teach or disclose any of these two limitations.

With respect to claim 81, the examiner argues that he believes WebWriter teaches instructions to decide about exclusions of components. He recites " ...where each page can contain both static content that is always present and computed content that is generated on the fly ... ". In applicant's reading, this portion does not reveal an operation of excluding a complete dynamic area from a page. It just reveals that the content of dynamic areas is computed at run-time. In contrast, the portion cited in the response makes clear that the WebWriter page generator processes each dynamic area on a template page by executing the associated script and has no ability to exclude a script of a dynamic area from this process. With respect to examiner's

argument of nested components, applicant submits that components are denoted on document templates as required by claim 74, but document templates are not components in the sense of the claim and therefore a dynamic area on a document template cannot be interpreted as prior art for nested components. See section A.ix.

With regard to claim 83-85, the examiner writes that applicant rehashes arguments as previously addressed above. However, claim 83 requires an editable view that includes and excludes components, claim 84 requires more components on page that on the template, and claim 85 requires multiple instance of components. All these are features that have not yet been discussed with other claims.

With regard to claim 88, the examiner refers to the reasoning applied to claim 81. Claim 88 however deals with multiple instances of a component while claim 81 with excluding components.

For claim 91, the examiner states that applicant is arguing for an unclaimed merit of distinction and cites "second browsers". In fact, the claim recites a "second window". WebWriter's editor does not disclose a second window but displays information on elements in the main window. In contrast, Claim 91 requires "second window".

With respect to claims 92-96, the examiner claims that applicant's arguments disclose previously addressed limitations. Applicant, however, submits that these claims and the base claim 90 use substantially different claim language and should therefore considered separately. Claim 96 introduces the limitation "wherein links contained in said document stay functional" which was not discussed anywhere else.

With respect to claim 115, the examiner refers to dynamic areas of editor as disclosed by Web Writer on page 2. In applicant's reading of page 2, no "*instructions for passing information to the editor"* are disclosed. In applicant's reading, page 2 left column under the headline *"The WebWriter Application Model"* last paragraph *"specification of locations within the page, called dynamic areas, where computed content should be inserted at* ***runtime*** *to produce the final page that should be displayed to the user"* (emphasis added) just discloses instructions "*to produce the final page that should be displayed to the user"* but no instructions to pass information to the editor. In addition, these instructions are used at runtime after editing and therefore cannot pass information to the editor.

With respect to claims 116 -119, the examiner states that applicant's arguments disclose previously addressed limitations. Claims 116-118, however, include additional limitations on "said information," a term introduced in the base claim 115 using the claim language "passing information to the editor". Limitations of "said information" have not been discussed before. Claim 119 introduces components having "instructions that display additional editing features". So far editing features have been discussed as part of components.

With respect to claim 121, the examiner introduces various elements including HTML tags and interprets these as applicant's term extension. The claim however requires an "extension for enabling editing of an attribute value of the components". The items cited by the examiner appear unrelated to editing of attribute values of dynamic areas and therefore are not an extension as required by claim 121.

Claim 122 requires "said extension enables display of a page for editing the attribute values of the components." The items the examiner interprets as extension are unrelated to display of a page for editing the attribute values and therefore the examiner reasoning is not applicable.

With respect to claim 123 the examiner writes "examiner interprets the tag name …". Applicant submits that the tag name is a well defined term in web technology and cannot be interpreted differently.

With respect to claims 126 and 127, the examiner states that applicant is rehashing previous arguments. With respect to claim 126, applicant notes that the limitation "the running step and the displaying step are repeated after initiating the modification function." clearly recites the feature of running during editing.

Although the feature of running during editing has already been discussed, the examiner's argument on page 23, last paragraph, is that this feature is not correctly recited in claim 1. Since claim 126 uses much different claim language, including details about running during editing by stating "the running step and the displaying step are repeated after initiating the modification function."

With respect to claim 127 applicant notes that the identifying step is only present in base claim 125 and limitations involving the identifying step as introduced in claim 127 have not been discussed elsewhere.

F. Response to Arguments given in the Interview on September 10th, 2009

In the interview, the examiner gave more concrete reasoning with respect to claim 1. He reasoned that the WebWriter prior art shows all the elements recited in the claim, namely, the WebWriter page generator as the claimed document generator, and the WebWriter editor as the claimed editor. However, applicant submits that the examiner has failed to consider the structural connections and limitations of these elements which are expressly recited in the claim.

The document generator element recites "*generating the generated documents, said generated documents including* ***additional editing features*** *for interpretation by the browser program*". In contrast, the WebWriter page generator is a program separate and apart from the editor, and it is not used at all during editing, but only after editing, and therefore its generated pages do not contain editing features. The following citations support this: "***Once*** *a Web Writer application is created and* ***saved*** *to disk it can be* ***run***" p. 6, col. 2 of WebWriter I, in the section entitled "Running the Application" (emphasis added) and **"When** a WebWriter application is **running**, each new page is assembled by the **WebWriter Page Generator**, ***another*** server based CGI C++ **program**..." first paragraph of the "WebWriter Page Generator" section on page 9 (emphasis added) .

In contrast, the WebWriter editor itself generates pages having editing features, but not the WebWriter page generator.

The claim further requires a data flow from the page generator into the editor by requiring the editor to operate on the pages generated by the page generator and to operate via the editing features.

The examiner further cited text from the reference: "*WebWriter then adds a placeholder for the output area [which] will be replaced* ***at runtime*** *by the output of a script . . . .*" (*see* p. 4, 2nd ¶, under heading "Placing Dynamic Areas", emphasis added), and stated that he interprets the replacement at runtime as being an editing operation. Applicant totally disagrees. The WebWriter page generator transforms a page template into a generated page for display by a web browser, thereby replacing dynamic areas with the output of scripts. This process is usually called template expansion or running of a dynamic web application. It is never interpreted as an editing process, because it does not permanently modify the page template but just temporarily

transforms the page template into a page for display, and later discards the page. Also, in editing, there is usually an explicit user interaction that requests a modification be made. The replacement of dynamic areas in the WebWriter page generator , however, is a totally automatic process taking place as an implicit step in executing an application. Finally, in the context of the claim, editing means editing of software applications, and that is different from transforming a page template into a final page.

In addition, even if the replacement at runtime operation was interpreted as editing, this does not teach or suggest that the WebWriter page generator generates pages having editing features as claimed, nor does it teach that the WebWriter editor operates via the editing features.

II. CONCLUSION

For all the foregoing reasons, applicant submits that the claims are in condition for allowance, and the Examiner's reconsideration to that end is respectfully solicited. The Examiner is encouraged to telephone the undersigned should additional issues remain.

Respectfully submitted,

Date: October 1. 2009 By: /Richard A. Nebb/
Richard A. Nebb
Reg. No. 33,540

VIERRA MAGEN MARCUS & DENIRO LLP
575 Market Street, Suite 2500
San Francisco, California 94105
Telephone: 415.369.9660
Facsimile: 415.369.9665